Homework

Visit course website, click on Resources and then on Code Style Guidelines. Study

4.2 Keep methods short
4.3 Use statement-comments …
4.4 Use returns to simplify method structure
4.6 Declare local variables close to first use …

Scope of local variables

```java
/** Return middle value of b, c, d (no ordering assumed) */
public static int middle(int b, int c, int d) {
    if (b > c) {
        int temp = b;
        b = c;
        c = temp;
    } // { b <= c }
    if (d <= b) {
        return b;
    } // { b < d and b <= c }
    return Math.min(c, d);
}
```

Scope of local variable (where it can be used): from its declaration to the end of the block in which it is declared.

References to text and JavaSummary.pptx

- Local variable: variable declared in a method body B.10–B.11 slide 45
- Inside-out rule, bottom-up/overriding rule C.15 slide 31-32 and consequences thereof slide 45
- Use of this B.10 slide 23-24 and super C.15 slide 28, 33
- Constructors in a subclass C.9–C.10 slide 24-29
- First statement of a constructor body must be a call on another constructor —if not Java puts in super(); C.10 slide 29

Local variables

```java
/** Return middle value of b, c, d (no ordering assumed) */
public static int middle(int b, int c, int d) {
    int temp = b;
    b = c;
    c = temp;
} // { b <= c }
    if (d <= b) {
        return b;
    } // { b < d and b <= c }
    return Math.min(c, d);
}
```

Parameter: variable declared in () of method header

<table>
<thead>
<tr>
<th>b</th>
<th>c</th>
<th>d</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>6</td>
<td>7</td>
<td>?</td>
</tr>
</tbody>
</table>

All parameters and local variables are created when a call is executed, before the method body is executed. They are destroyed when method body terminates.

Principle: declaration placement

```java
/** Return middle value of b, c, d (no ordering assumed) */
public static int middle(int b, int c, int d) {
    int temp;
    if (b > c) {
        temp = b;
        b = c;
        c = temp;
    } // { b <= c }
    if (d <= b) {
        return b;
    } // { b < d and b <= c }
    return Math.min(c, d);
}
```

Not good! No need for reader to know about temp except when reading the then-part of the if-statement

Principle: Declare a local variable as close to its first use as possible.
**Assertions promote understanding**

```java
/** Return middle value of b, c, d (no ordering assumed) */
public static int middle(int b, int c, int d) {
    if (b > c) {
        int temp = b;
        b = c;
        c = temp;
    }
    // { b <= c }
    if (d <= b) {
        return b;
    } // { b < d and b <= c }
    return Math.min(c, d);
}
```

**Assertion:** Asserting that $b \leq c$ at this point. Helps reader understand code below.

---

**Bottom-up/overriding rule**

Which method `toString()` is called by:

```java
c.toString() ?
```

**Overriding rule or bottom-up rule:** To find out which is used, start at the bottom of the object and search upward until a matching one is found.

---

**Inside-out rule**

*Inside-out rule:* Code in a construct can reference names declared in that construct, as well as names that appear in enclosing constructs. (If name is declared twice, the closer one prevails.)

**Parameters participate in inside-out rule**

To see what declaration each reference $v, w, x, y, z$ refers to, look in inside-out fashion:
1. then-block
2. method body
3. parameter list
4. fields
5. static variables

---

**Static items participate in inside-out rule**

In object `Person@0`,

```java
setN(String name) {
    n = name;
}
```

In object `Person@1`,

```java
setN(String name) {
    n = name;
}
```

---

**A solution: use `this`**

Memorize: Within an object, `this` evaluates to the name of the object.

In object `Person@0`,

```java
setN(String name) {
    this.n = name;
}
```

In object `Person@1`,

```java
setN(String name) {
    this.n = name;
}
```

---

**Box for class C**

| C@0  | v    | m(int y) {
|      |      |   int x;
|      |      |   if (...) {
|      |      |     int z;
|      |      |     ... v ... w ...
|      |      |     ... x ... y ... z
|      |      |   };
|      |      | w    |

Variables: static $w$
field $v$
parameter $y$
local variables $x$ and $z$

To see what declaration each reference $v, w, x, y, z$ refers to, look in inside-out fashion:
1. then-block
2. method body
3. parameter list
4. fields
5. static variables

---

**Person@0.n is this variable**
Within a subclass object, `super` refers to the partition above the one that contains `super`.

Because of the keyword `super`, this calls `toString` in the Object partition.

```java
public class Time
private int hr; // hour of day, 0..23
private int min; // minute of hour, 0..59
/** Constructor: instance with h hours and m minutes */
public Time(int h, int m) {
    this(m / 60, m % 60);
}
/** Constructor: instance with m minutes */
public Time(int m) {
    hr = m / 60;
    min = m % 60;
}
...
```

Use `this` (instead of `Time`) to call another constructor in the class. Must be first statement in constructor body!

**Without OO …**

```java
public double getCompensation(...) {
    if (worker is an executive) {
        // ... }
    else if (worker is part time) {
        // ... }
    else if (worker is temporary) {
        // ... }
    else ... 

    // OO eliminates need for many of these long, convoluted methods, which are hard to maintain.
    // Instead, each subclass has its own getCompensation.
    // End up with many more methods, which are usually very short
```
Principle: initialize superclass fields first

/** Constructor: employee with name n, year hired d, salary s */
public Employee(String n, int d, double s)

/** Constructor: executive with name n, year hired d, salary of $50,000, bonus b */
public Executive(String n, int d, double b)

Principle: In subclass constructor, fill in the superclass fields first
How to do that if they are private?

Call constructor in superclass

/** Constructor: employee with name n, year hired d, salary s */
public Employee(String n, int d, double s) {
    super(Employee(n, d, 50000));
    bonus = b;
}

Principle: initialize superclass fields first

/** Constructor: an instance with … */
public C(…) {
    super();
    S0;
    S1;
    …
}

Java syntax: First statement of any constructor you write must be a call on another constructor
this( … ); or super( … );

If you don’t put one in, Java silently inserts this one:
super();